High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development

DOE NNSA
Nathan DeBardeleben, LANL[1]
James Laros, SNL[2]

DOD ACS Research Program
John Daly, CEC

DOE Office of Science
Stephen Scott, ORNL[3]
Christian Engelmann, ORNL[3]

DOD DARPA
Bill Harrod, IPTO[4]

Executive Summary 2
Background and Introduction 3
Purpose and Scope 6
Frequently Used Terms 7
Historical Perspective 11
Key Areas for Future Research, Development, and Standards Work 15
- Thrust #1: Theoretical Foundations 16
- Thrust #2: Enabling Infrastructure 17
- Thrust #3: Fault Prediction & Detection 20
- Thrust #4: Monitoring & Control 22
- Thrust #5: End-to-End Data Integrity 25

Conclusion 27
References 28

Collaborators: Arijit Biswas (Intel), James Brandt (SNL), Anne Gentile (SNL), Sanjay Kale (UIUC), Sarah Michalak (LANL), Henry Newman (Instrumental), Jon Stearley (SNL)

---

[1] This paper has been approved for release under LA-UR-10-00030
[2] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Executive Summary

High-end computing (HEC) is requisite for solving our nation's most important scientific and engineering problems, and has become increasingly vital to the mission of the national security community [1]. As the scale and complexity of HEC systems continues to grow, the impact of faults and failures will make it increasingly difficult to accomplish productive work using traditional means of fault-tolerance [2, 3]. Further, the challenges of integrating large complex heterogeneous systems are increasing to the point where the "stabilization" period consumes a significant portion of the lifetime of those systems [4, 5]. As a consequence of these two markedly disturbing trends, it will be necessary for the HEC community to identify innovative means for efficiently and affordably performing productive work on systems encountering frequent, persistent and erratic errors, many of which will be undetectable by existing system monitoring solutions. Resilience meets these profoundly daunting and ever increasing challenges. To ensure the continued viability of the largest, most powerful, leading edge computing systems will require standards based solutions. These solutions must efficiently and dynamically guard and preserve information, computation and data movement in the presence of faults and failures arising from complex system interactions and dependencies among platform hardware and software components, the system workload, and the physical environment.

> **The goal of HEC resilience is to enable effective and resource-efficient use of computing systems at extreme scale in the presence of system degradations and failures.**

At the highest level, high-end computing is the process by which data is transformed into information through computation. Resilience facilitates this critical transformation process by accepting that the underlying hardware and software that comprises a system will be unreliable. In order to succeed, resilience assumes a new perspective in which uncertainty about the state of the system plays an important role in managing that system. Resources traditionally focused on maintaining a known and desirable system state are instead focused on end-to-end fidelity of data, computation, and data movement. Resilience is concerned with reliability of information in lieu of, or even at the expense of, reliability of the system. This novel approach to fault-tolerance is necessary to address the two-fold challenge of decreasing system reliability, because of increasing scale, and decreasing certainty about the operational state of the system, because of increasing complexity. The resilience community proposes to address these challenges in five focused but overlapping thrust areas: *theoretical foundations*, *enabling infrastructure*, *fault prediction & detection*, *monitoring & control*, and *end-to-end data integrity*. To manage this prodigious scope, a successful program of resilience research will require coordinated, multi-disciplinary undertakings in each of these thrusts areas. This requirement forms the justification of a call for a national effort in resilience.

Background and Introduction

Recent trends in high-end computing (HEC) system design have clearly indicated future increases in performance, in excess of those resulting from improvements in single-processor performance; will be achieved through corresponding increases in system scale, i.e., using a significantly larger component count. As the raw computational performance of the world's fastest HEC systems increases towards the next-generation exascale capability, the number of computational, networking, and storage components will grow enormously. To put this in today's terms, presently the largest petaflop-plus machine [6], Jaguar from Oak Ridge National Laboratory with a peak speed of 2.33 petaflops, has over 250,000 cores [7]. Similarly, the first petaflop supercomputer, Roadrunner, deployed at Los Alamos National Laboratory, has over 122,000 cores with a peak speed of 1.38 petaflops. Furthermore, it is anticipated that an exascale machine will reach the 100-million core and perhaps even 1-billion cores [8].

With only very few exceptions, the reliability and availability of recent HEC systems have been lower in comparison to the same deployment phase of their predecessors. Based on personal communication with various HEC center personnel and based on publicly available statistics (Table 1), the overall system availability of currently operational HEC systems is roughly above 96% and below 99%, not including the time spent on checkpointing and recovery. It is important to note that the availability of a system is a measure of how often components are in a usable state. The productive runtime is represented as runtime efficiency and depicts the amount of useful cycles that are applied to solving a problem and does not include checkpointing. The numbers combine as a product to produce the faction of useful cycles obtains on a given system.

In some cases, the overall system mean time between failures (SMTBF) is under two hours. Previous work similarly suggests a system mean time to failure (SMTTF) constraint of 5-6 hours, or 4 failures per day, for current HEC systems [9]. The most common reported sources of failure in large-scale distributed systems are processor malfunction, memory errors, and storage failures.

Installed	System	Processors	SMTBF	Measured	Source
2000	ASCI White	8,192	40.0h	2002	[10]
2001	PSC Lemieux	3,016	9.7h	2004	[11]
2002	NERSC Seaborg	6,656	351.0h	2007	[12]
2002	ASCI Q	8,192	6.5h	2002	[13]
2003	Google	15,000	1.2h	2004	[14]
2006	Blue Gene/L	131,072	147.8h	2006	[15]

Table 1. Publicly Available Past and Current HEC System Reliability Statistics

In general, parallel applications run faster (lower wall clock time) in comparison to their single threaded counterparts. However, when substantially increasing the number of nodes located within an HEC system and assuming theoretical linear scalability for applications, application completion times do not necessarily decrease proportionally. In fact, it has been shown that (Figure 1) [16] the opposite is true – while application completion time initially decreases as nodes are added, at some critical point this value begins to rise substantially due to the increased likelihood of reliability issues stemming from the additional computational units.

As a result, HEC centers may artificially set allowable job run time for their HEC systems to very low numbers in order to require a scientific application to store intermediate results, essentially a forced checkpoint, as insurance against lost computation time on long-running jobs. However, this forced checkpoint itself wastes valuable computation time and resources as this checkpoint time does not produce additional scientific results yet does consume computation time and other resources.

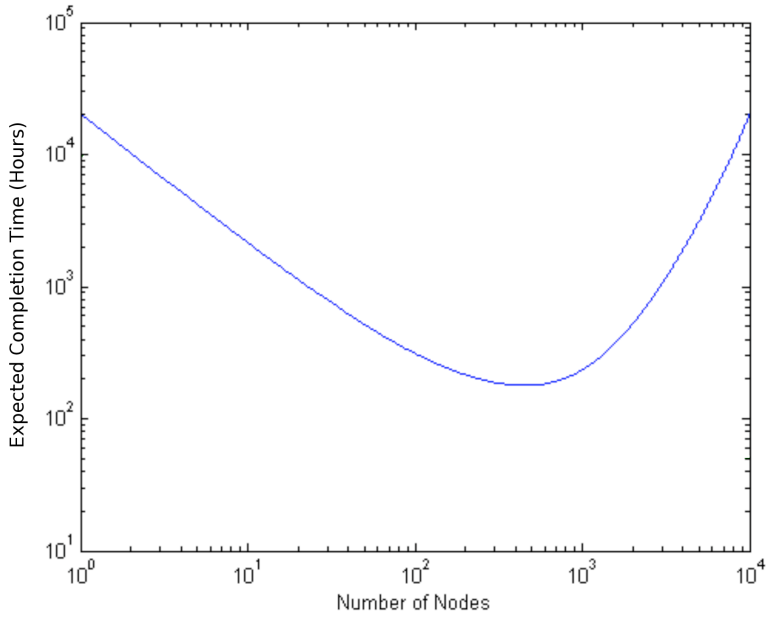

Figure 1. Application Completion Time vs. Number of Nodes

In contrast, the demand for HEC system availability has risen dramatically with the recent trend toward capability computing, which drives the race for scientific discovery by running applications on the fastest machines available while desiring significant amounts of time (weeks, months, or more) without interruption. Looking ahead, the expected growth in HEC system scale poses significant challenge for system and application fault resilience.

A recent study [17] estimates the SMTBF for a next-generation extreme-scale HEC system. Extrapolating from current performance, scale, and overall system mean-time to failure (SMTTF), it suggests that the SMTBF may fall to only 1.25 hours of useful computation on a 1 PFlop/s (10^{15} floating point operations per second) system. This study also estimates the overhead of the current state-of-the-art fault-tolerance strategy, checkpoint/restart, for a 1 PFlop/s system, showing that a computational job that could complete in 100 hours in a failure-free environment will actually take 251 hours. What this analysis implies is startling: more than 60% of the processing power (and investment) on these extreme-scale HEC systems may be lost due to the overhead of dealing with reliability issues, unless something happens to drastically change the current course of system reliability.

This paper takes an in depth look at the current state of the HEC resilience community with the aim of identifying key research areas where investment should be made to reach the goal of more resilient HEC systems of tomorrow. We begin by outlining some frequently used terms and then take a detailed look at the history of reliability, fault-tolerance, and resilience in the HEC arena. We then outline five key research thrusts which are interrelated and contain a great deal of overlap. This necessitates a global view of the problem of resilience to best assign resources and calls for a government-lead initiative in this area. We identify some existing research in these thrusts and point to serious gaps that impede the national HEC goals. Finally, we conclude with a look the level of impact action or inaction will have, positive or negative, on the future of HEC.

Purpose and Scope

The purpose of this document is to identify the critical research and development areas required to achieve the goal of resilience within High-End Computing (HEC). We assume the need for HEC systems will continue to outpace their availability. Further, left unchecked, these systems will ultimately become unusable due to hard and soft errors, and could potentially result in errors in the data that might propagate to incorrect decisions or actions. Usability will be threatened and ultimately we question whether the high costs of providing resilience in both hardware and software will provide acceptable returns.

This document does not address breakthrough reliability technologies that would fundamentally change the resilience field. Per component hardware reliability has been remarkably consistent over the history of the HEC industry, so we assume an evolution of hardware reliability and an incremental awareness of application resilience. While several HEC vendors are looking to address reliability at the hardware level, the costs are proving to be staggeringly high in both money and power. This document assumes that while new reliability technologies will continue to be integrated into next generation HEC systems, these technologies will not be sufficiently disruptive to precipitate radical or wholesale changes in the resilience needs of platforms and applications.

Frequently Used Terms

The following is a list of terms and associated definitions that occur frequently in standard literature pertaining to computer system reliability:

Algorithm-Based Fault Tolerance (ABFT) – Fault tolerance technique for computational algorithms designed to either ignore failures, and still deliver an acceptable result, or to implicitly recover at low cost using computation. A major requirement is that the underlying system (hardware and software) supports degraded operation (see *Degraded Modes*).

AMTTFE – "Application Mean Time to Fatal Error" is a measure of the likelihood that an application running on a particular set of platform components will run without error or interruption for the duration of its runtime.

Application – Codes run by users that apply programming models to implement numerical methods with the intended function of transforming data and/or solving equations to give accurate information about real phenomenon.

Application Monitoring – The act of monitoring the progress of an application. Different fidelity can be achieved through the combination of internal and external application monitoring. Application monitoring is usually used to identify if an application is stalled, having problems making forward progress at an acceptable rate, and/or how far along in its computation it has progressed.

Byzantine Fault Tolerance – The process of masking any kind of failure, including those caused by soft errors. It expands traditional fault tolerance to include malicious failures, such as the Byzantine generals problem [18].

Checkpoint/Restart (C/R) – A fault-tolerance technique whereby application/system state is saved in intervals to persistent storage, which is typically a network shared file system with redundancy for robustness. Upon failure, the previously saved state is restored, typically by restarting failed applications. Progress between the last checkpoint and the time of failure is lost. Global C/R coordination is required to ensure consistency. The efficiency of C/R depends on checkpoint and restart duration, interval, and MTTF. Many applications checkpoint after computational phases, such as a set of iterations, in order to be able to restart from an intermediate result. In some specific situations, system software solutions offer transparent mechanisms. C/R has been the state of practice for HEC fault tolerance for decades. See also *Checkpoint/Restart, Diskless*.

Checkpoint/Restart, Diskless – A subset of C/R techniques that save state in memory rather than persistent long-term storage. This usually has tradeoffs with respect to latency, bandwidth, capacity, contention and most importantly reliability as the memory often resides in a compute resource susceptible to failure where the checkpoint could potentially be unavailable for retrieval. Commonly, techniques utilize some type of parity or raid-like mechanism to ensure correctness and redundancy and utilize a remote node's memory, allocated for this purpose.

Degraded Modes – System states in which operation is abnormal or defective but does not terminate.

Dependent Failure – An event in which one or more components lose the ability to perform their intended function because of the failure of some other component on which they are dependent for correct operation.

ECC – An "Error Correcting Code" is redundant data that is added to a data stream to allow the recipient of that data to detect and correct some number of erroneous bits, where the number of correctable bits is determined by the method and the degree of redundancy.

Error – The condition created by a fault that causes system hardware or software to stop performing its intended function or to perform it incorrectly and may lead to a failure (see *Fault* and *Failure*).

Error Detection – The process of accurately determining when a system (i.e., platform or application) has ceased to perform its intended function or is performing it incorrectly.

Error Latency – The time elapsed between the point at which the system stops performing its intended function correctly and ceases functioning entirely (see *Error* and *Failure*).

Error Propagation – The process by which one or more system errors degrade the state of the system until that system ceases to operate (see *Error* and *Failure*).

Fail Silent Violation – Instance of faults that leave a system operating in an abnormal or defective manner such that it is creating errors.

Failure – The point in time at which the system hardware or software ceases functioning as a consequence of one or more errors (see *Fault* and *Error*).

Fault – An abnormal condition or defect in system hardware or software that *may* lead to a failure (see *Error* and *Failure*).

Fault, Intermittent – A system fault resulting inconsistently or erratically from conditions on the system that may not be repeatable.

Fault, Permanent – A system fault consistently resulting from conditions on the system that are repeatable.

Fault, Transient – A system fault resulting inconsistently or erratically from conditions in the environment that are not repeatable.

Fault Activation – The process by which an abnormal condition or defect in the system causes that system to begin performing its intended function incorrectly or not at all (see *Fault* and *Error*).

Fault Prediction – The process of accurately predicting the occurrence of an abnormal condition or system defect before it causes an error condition in which they system is no longer function correctly.

Fault Tolerance – The ability of a system to continue performing its intended function properly in the face of transient, intermittent, and permanent faults.

Fault-Tolerant Message Passing Environments – A specific class of message passing programming model implementations that provide high availability for a distributed application and is responsible for messaging and process/task management. Applications are able to adapt at runtime as long as this software layer is able to detect, isolate, and signal failures.

Fidelity – A notion of how accurate an application or platform is. Often used as "tunable fidelity" in the sense of a configurable amount of accuracy presumably in exchange for something (power, performance, monetary cost, etc).

Hard Error – An error resulting in the permanent malfunction of a device.

Log-Based Failure Analysis and Prediction – A technique used for prediction and root cause identification usually utilizing statistical methods. Most work relies on event logs made public via the USENIX Computer Failure Data Repository at http://cfdr.usenix.org.

Message Logging (ML) – A technique that saves all parallel application messages in a log. A failed application part is restarted and its input is replayed to restore state. Surviving parts are not affected. ML is only applicable to deterministic applications. It may be combined with C/R to avoid playback from start and to enable uncoordinated C/R protocol using the message log for consistency. The efficiency of ML depends on the introduced runtime overhead, the MTTF, and if any C/R is used.

Middleware – The set of software that interfaces between the user applications and the underlying operating system and hardware. Examples include runtime systems, libraries, and device drivers.

Mis-corrected Error – This is an error that is one of the potential results of SDC. In this case the ECC correct what it thinks is an error in the data, incorrectly.

Multi-Component Failure – An event in which an error in a single component of hardware or software is responsible for the failure of multiple other components in the system.

Performability – A measure of the rate of computation work performed on a system that includes consideration of both the performance and reliability of the system.

Platform – The integrated set of hardware components (nodes, interconnect, memory, etc) and software infrastructure (operating system, libraries, compilers, etc) designed with the intended function of running one or more applications.

Proactive Fault Tolerance – keeps applications alive by avoiding failures through preventative measures, such as rejuvenating the state of the system to remove latent faults that could cause an error later or migrating application parts away from components that are "about to fail" using early failure indications.

Programming Models or Programming Paradigms – A fundamental "style" of computer programming that differs based on the abstractions used to represent programs in that paradigm. Well-known abstractions include objects, functions, and variables.

Provenance – Refers to the history or source of data in a computation. In the context of resilience, knowledge of provenance is necessary for the recovery of a correct system state

following an error by providing a record of hardware and software components with the potential to have been corrupted as the error propagated.

QoS – "Quality of Service" is a measure of the ability of a system to maintain an acceptable level of service where that level of service may be different for different platform, applications, workloads, or users.

Reliability – A measure of "the ability of a system or component to perform its required functions under stated conditions for a specified period of time." [19]

Recovery Oriented Computing – An approach to fault tolerance focused on minimizing error latencies (i.e., "fail quickly") and getting things running again quickly after a failure. This approach is in contrast to resilience that attempts to keep applications running without the need for stopping and restarting.

Resilience – The ability of a system to keep applications running and maintain an acceptable level of service in the face of transient, intermittent, and permanent faults.

Silent Data Corruption (SDC) – "SDC occurs when incorrect data is delivered by a computing system to the user without any error being logged." [20]

Single Fault Violation – Instance of faults that occur simultaneously in multiple locations as a result of a single event or an interaction of multiple events.

SMTBF – "System Mean Time Between Failures" is a measure of the reliability of a system that is the operational time of a system during some period divided by the number of failures that have occurred during that period.

Soft Error – A change in the "state of a logic device" that "does not reflect a permanent malfunction of the device." [21]

Soft-Error Fault Tolerance – A type of fault tolerance that focuses on techniques for masking software component failures or data corruption, as opposed to handling permanent device malfunctions and hard fails (see *Soft Error* and *Hard Error*). These error types include transient failures that are only active as long as the faulty software component is not rebooted or the corrupt data location is not overwritten.

System – The combination of a given application or set of application (i.e., an application workload) and a given platform on which they are running.

Undetected Error – This is an error that is one of the potential results of SDC. The ECC and the data packet have both have errors that when combined are not detected and passed to the next device.

Historical Perspective

In the following, the evolution of resilience technologies for HEC over the last 3 decades is summarized, examples are given, and important milestones are briefly described. The current state of research and practice is shortly illustrated and future demands are discussed.

Evolution of Resilience Technologies

Checkpoint/restart (C/R) has been the state of practice for HEC fault tolerance for decades. For example, the 1982 4-processor Cray X-MP [22] had optional fast solid-state drive (SSD) checkpoint storage. The 1988 follow-on 8-processor Cray Y-MP and the 1985 6-processor IBM 3090 [23] had the same feature. As processor counts increased, systems had more distributed properties. C/R latency grew with the move from direct-attached storage (DAS) to network-attached storage (NAS). For example, the 1992 2048-processor Intel Paragon XP/S [24] had compute nodes without storage and dedicated I/O nodes with DAS or NAS. With the advent of cluster computing in the mid-to-late 1990s, compute-node storage became popular. For example, the 2000 8,192-processor ASCI White was an off-the-shelf cluster composed of 16-processor IBM RS/6000 [25] nodes, each with 20GB DAS. Although compute-node storage permits checkpoint caching, it was rarely used in practice. The concern over disk reliability forced DAS out of compute nodes in the early 2000s. For example, since 2004, the IBM Blue Gene [26] and Cray XT [27] series embrace the massively parallel processing (MPP) architecture of the early 1990s with I/O nodes and NAS.

Application-level C/R has been the predominant HEC fault tolerance method for decades. Many applications checkpoint after computational phases to be able to restart from an intermediate result, like the 1996-now Community Climate System Model (CCSM) [28]. Since 2004, the C3 [29] C/R toolkit offers compiler-assisted transformation of any MPI application to become self-checkpointing and self-restartable.

System-level C/R first appeared in 1995 with libckpt [30], a portable Unix C/R tool with transparent, incremental, forked and user-directed checkpointing of processor state, process stack and data. In 1996, CoCheck [31] added coordination for MPI applications to achieve consistency by assuring global recovery lines. In 1997, Condor [32] added checkpointing of shared library code/data, open files state, signal handlers and pending signals. Since 2003, Berkeley Lab Checkpoint Restart (BLCR) [33, 34, 35] offers transparent process C/R in Linux that additionally includes process identifiers. BLCR is integrated with LAM/MPI [36, 37] and Open MPI [38] for transparent C/R of MPI applications.

The concept of **diskless C/R** was first developed in 1997 [30]. It offered transparent, incremental, forked and user-directed checkpointing of MPI applications to the memory of dedicated or spare nodes with parity encoding for robustness. A 1998 prototype [39] added scalable checkpointing to neighbor compute nodes. In 2004, transparent diskless C/R with checkpoint replication for robustness was added to the Charm++/AMPI [40] framework, which also supported dynamic load balancing. The Scalable C/R (SCR) library

[41], released in 2009, offers local and neighbor diskless C/R with parity encoding or replication.

Fault-tolerant message passing was first available with the 1993 PVM 3 [42], which survived n−1 process failures and offered dynamic reconfiguration. Starfish [43], developed in 1999, was the first fault-tolerant MPI. Its scalability was limited due to the use of the Ensemble [44] group communication system. The 2001 FT-MPI [45] provided more scalability. It also extended the MPI specification with fault tolerance features that are currently discussed for the MPI-3 standard [46].

In 1992, Manetho [47] offered the first **message logging (ML)** solution with antecedence graph maintenance, uncoordinated checkpointing, and sender-based ML. Its recovery scheme offered limited rollback, fast output commit and low overhead. The 1999 Egida [48] provided ML for MPI applications with a specification language to automatically synthesize a ML protocol implementation. In 2006, MPICH-V [49] added two advanced ML protocols: (1) Chandy Lamport [50] and (2) a more scalable Blocking Chandy Lamport.

Algorithm-based fault tolerance (ABFT) first emerged in 1984 for matrix operations in systolic arrays [51]. It relied on computing encoded matrices to allow for "offline" recovery, i.e., after the computation. Re-emerged in 2006, an extension [52] offered "online" recovery, i.e., during the computation. Related research in 2005 focused on "naturally" fault-tolerant algorithms [53] based on a composition of local algorithms, e.g., chaotic relaxation [54]. A 2007 extension [55] offered ABFT for parabolic heat transfer problems that overcomes the issue of irreversibility in time by numerically reconstructing lost data.

Developed in 2007 and 2008, two prototypes [56, 57] offered a new **proactive fault tolerance** approach through migration of application parts (virtual machines or processes) away from components that are "about to fail" using simple threshold triggers on environmental monitoring sensors.

Log-based failure analysis and prediction has been a very recent effort. For example, the 2007 hPREFECTs failure prediction framework [58] explored log entry correlations with a more than 76% accuracy in offline prediction and more than 70% accuracy in online prediction. Also, in 2008, Sisyphus [59] offered a classification scheme for syslog messages. It is able to localize 50% of faults with 75% precision, corresponding to an excellent false positive rate of 0.05%.

A 2008 **Byzantine fault tolerance** solution [60] for HEC Grids offered an automated mechanism for running parallel applications in a replicated fashion with reasoning over output validity.

For **soft-error** resilience parity memory was deployed in HEC systems until the early 1980s, such as in the 1977 Cray-1 [61], and then replaced by more resilient single-error correction (SEC) double-error detection (DED) error correcting code (ECC) memory, e.g., in the Cray X-MP. Reports of bit-flips in onboard caches in the 2002 ASCI Q system [62] forced processor vendors to move from parity to ECC for caches and registers, such as in the AMD Opteron [63] deployed in Cray's XT series.

Current State

With a few exceptions, today's HEC systems assure fault tolerance in the same way since the early 1990s. Application-level C/R with NAS is the predominant HEC fault tolerance method for hard errors, and ECC throughout the memory hierarchy is the prevalent HEC soft error resilience technique. System-level C/R is employed at a few HEC centers, e.g., using BLCR. However, none of the current petascale HEC centers support system-level C/R. Diskless C/R has only recently begun being used in production HEC systems at LLNL (SCR library). Message logging, algorithm-based fault tolerance, proactive fault tolerance and Byzantine fault tolerance are not available in production HEC systems. Log-based failure analysis and prediction is sparsely used in root cause analysis on large-scale machines. The current state of practice in HEC resilience is largely disconnected from the described current state of research.

Recent efforts providing an insight into HEC resilience showed that some HEC system failures can be anticipated by detecting deteriorating system health through hardware monitoring (fan speeds, temperatures, disks error logs) [64]. Other work focused on capturing the availability of large-scale systems using combinatorial and Markov models and on comparing these results with statistics from large-scale U.S. Department of Energy (DOE) installations [65, 66]. Unfortunately the health data collection and processing outlined in these studies will not perform efficiently on large-scale HEC systems, so the existing work will need substantial research investment to leverage it effectively.

In contrast to HEC platforms used in research, large-scale commercial IT installations experience interpolated fault rates similar to those expected of next-generation HEC systems, e.g, Google's SMTTF of 1.2 hours (Table 1). However, their fault-tolerant middleware hides these failures from customers, while successfully maintaining end user service provision [67].

Future Demands

As HEC systems continue to increase in size, their system mean time to failure (SMTTF) decreases dramatically due to increasing component counts resulting in more frequent system-wide interruptions [2, 68]. The solution, up until now, has been to increase component reliability. However, this is not sustainable. No vendor will/can build at a reasonable cost a component (processor, memory module, or network interface) that doesn't fail in 10-20 million hours (1370-2740 years) as needed for a system with 1,000,000 components and a desired MTTF of 12-24 hours.

Single event upsets (SEUs) and single event multiple upsets (SEMUs) [69], i.e., single and multiple bit flips caused by a single natural high-energy radiation strike [70, 71], are the primary source of soft errors in chip memory and logic [72, 73]. It is expected that SEU and SEMU vulnerability will further increase with shrinking nanometer technology [72, 73, 74]. Detectable unrecoverable errors (DUEs) caused by radiation-induced soft errors will very quickly become the predominant source of interruptions, while silent data corruption (SDC) incidents will initially be rare, but occurring, events [75, 72].

Recent investigations [76, 77] revealed that C/R efficiency, i.e., the ratio of useful vs. scheduled machine time, can be as high as 85% and as low as 55% on current-generation

HEC systems. With increasing error rates, increasing aggregate memory and not proportionally increasing I/O capabilities, traditional C/R via NAS is becoming less efficient and soon obsolete (at 50% efficiency).

Future-generation extreme-scale capability HEC systems will experience more frequent failures due to growing scale and shrinking nanometer technology. Resilience has to become an urgent priority to ensure that these leadership computing systems operate at an acceptable efficiency and productivity. In addition to exploring advanced resilience techniques, a sustained software research and development effort is needed for the full life cycle of resilience solutions as a software product, including basic research, prototyping and testing, technology integration and production deployment.

Key Areas for Future Research, Development, and Standards Work

The scope of the resilience challenge is immense in terms of both the number of tasks to be accomplished and the level of coordination across research communities required to accomplish them successfully. Figure 1 illustrates an approach to organizing the key challenge areas into five overlapping thrusts, each of which includes multiple research topics. The areas of overlap in the thrust diagram indicate points at which research activities will require some level of coordination in order to develop and standardize interfaces. In addition, the placement of the individual tasks within the thrusts, though mainly notional, is intended to convey the breadth and diversity of coordination required.

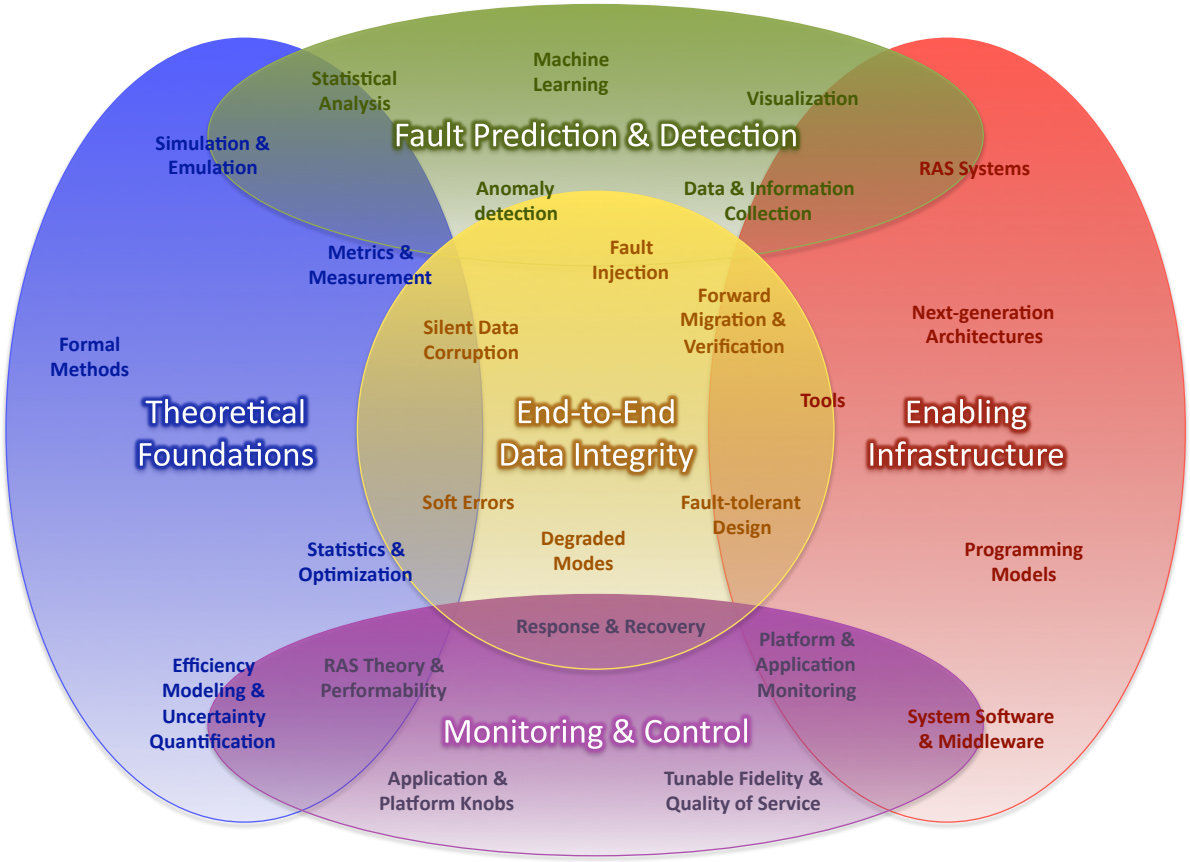

Figure 2. The five key areas of concern for resilience are overlapping and largely multi-disciplinary, thus motivating the need for an organized national effort to address the issues.

Notice that the first four thrusts are intentionally organized around the central thrust of end-to-end data integrity, which is the nucleus of HEC resilience. The sub-sections that follow give overviews of each of the thrust areas. They touch on some of the primary research topics and providing more information about the state of the art and gaps associated with the five thrusts.

Thrust #1: Theoretical Foundations

The topic area of theoretical foundations is primarily concerned with understanding "how" and "why" failures occur in HEC systems. This is a very broad category that is concerned with correctly describing the mechanisms and characteristics of failure from the level of system components and infrastructure down to the level of logic on the circuit. It addresses the compute platform as well as applications and must pay particular attention to interactions between platform and application in fault activation and error propagation.

Metrics & Measurement

Lord Kelvin is famously quoted as saying "If you cannot measure it, you cannot improve it." This universal truth necessitates a body of basic research in HEC resilience with respect to metrics and measurement. Standardized metrics are a critical enabler to improving HEC resilience. The absence of agreed definitions and metrics has obscured meaningful discussion of the issues involved and hindered their solution. Furthermore, fair and meaningful reliability comparisons between systems are impossible because of different hardware and software architectures, failure modes, and system health and failure reporting mechanisms. A reliability and availability metrics standard for HEC is needed as well as scalable, non-intrusive system health data collection and processing. Multiple groups have published on this topic recently [78, 79, 80]; the time is ripe to form a multi-agency, multi-vendor committee to publish a standard for measuring HEC resilience, and produce a reference implementation. Given proper priority, we believe this could be completed within two years, and would be a critical enabler for quantifiable improvements in HEC resilience.

Simulation & Emulation

One area of growing concern in the HEC community is the potential for increasing error latencies as the interval of time increases between fault activation, when a fault first compromises the end-to-end integrity of a calculation, and failure, when the error is detected. During this time the system is operating in a degraded state where performance and correctness of running calculations may be compromised, but the consequences of the error have not yet been detected. Unfortunately, the error latency is unmeasurable on any practical system [81] therefore must be modeled by simulation and emulation using methods such as fault-injection. However, the duration of the period of error propagation could exist on time scales ranging from nanoseconds to weeks or more for extreme scale systems, which makes the simulation problem very challenging.

Formal Methods

Formal methods are concerned with provability of correctness, particularly in the presence of Byzantine faults. In general, these sophisticated mathematical methods can be used to demonstrate that hardware and software architectures are resilient to error by design, and can be used in combination with knowledge of provenance as a rigorous guarantor of end-to-end data integrity. More research should be done in application of these types of methods to systems at extreme scale.

Statistics and Optimization

These two broad areas of research apply specifically to resilience in the context of creating a dynamically controlled feedback loop in which the system is monitored and corrective action is taken in response to the detection or prediction of errors. In particular, the resilience mechanism is a stochastic optimal control loop where the system inputs (i.e., faults) are governed by statistical processes and the governing equations and constraints are dictated by the power, performance, and reliability requirements of the system. More research is needed to understand how these principles are applied in real-time to complex systems at extreme scales.

Efficiency Modeling and Uncertainty Quantification

There are two broad requirements that fall under this research topic: (1) a theoretical or empirical model quantifying system efficiency in terms of power, performance, and reliability; and (2) a statistical model of quantifying uncertainty with regards these three system parameters (for example, see [82]) to enable analysis of underlying stochastic processes. Research in this area is critical, in conjunction with formal methods, for developing a QoS infrastructure for resilience that is able to provide tunable fidelity by guaranteeing a certain level of performance and correctness with a certain degree of certainty in the face of faults that are fundamentally statistical in nature.

Thrust #2: Enabling Infrastructure

The topic area of theoretical foundations is primarily concerned with understanding "how" and "why" failures occur in HEC systems. This is a very broad category that is concerned with correctly describing the mechanisms and characteristics of failure from the level of system components and infrastructure down to the level of logic on the circuit. It addresses the compute platform as well as applications and must pay particular attention to interactions between platform and application in fault activation and error propagation.

The topic area of enabling infrastructure is primarily concerned with the hardware and software components that are used in conjunction to create a more reliable HEC system. This is an extremely broad category that includes programming languages and models, RAS systems, system software, middleware (such as libraries), and tools in general. These components often need to cooperate to identify faults and maintain the stability of the system to keep the application running without interruption.

Programming Models and Languages

The burden of addressing resilience in the application level is high and it is not clear that the average domain scientist is best equipped to implement resilient codes. Just as programming languages and models have emerged to ease parallel programming complexities while providing performance, so too must we see new models to provide application resiliency. The DARPA HPCS effort has focused on creating languages that will provide high productivity solutions while still providing performance. We need similar efforts with respect to reliability solutions via languages and models.

Most HEC applications communicate via the Message Passing Interface (MPI). MPI is historically not tolerant to faults and there have been several solutions that looked to address this with varying levels of conformance to the standard. The MPI Forum is currently meeting to hopefully provide for a fault tolerance standard in the MPI 3.0 specification but currently users have no viable, standard, portable options. Other runtime systems, such as HARNESS [45], can enable fault tolerance in MPI applications, using FT-MPI [83], which survives the crash of up to n-1 processes in an n-process job; it is, however, the responsibility of the application to recover the data-structures on the crashed processor.

One successful solution in the research community has been Charm++ [84] that, through use of object-based-virtualization features, has enabled the development of various fault tolerance techniques for parallel applications on large systems. These applications can be written in Charm++ or in MPI, which is supported by Adaptive-MPI [85]. Charm++ can restart an application from a checkpoint, with the advantage that the restart can be done with any number of processors [86], since Charm++ decouples the application's data decomposition from the configuration of the underlying physical machine where it is executed. An extension of this scheme is a memory-based checkpointing [87] (see *Checkpointing, Diskless*), which is much faster because all the saved state is kept in remote processors' memories, not in disks. Clearly there exists a set of tradeoffs that must be managed.

Reliability, Availability and Serviceability (RAS) Systems

RAS systems make up a vital component in today's HEC systems. They are predominantly deployed on the larger systems that are most interested in high availability of the compute resources. RAS is a cross-cutting topic that is certainly part of the enabling infrastructure but we will discuss this topic in Thrust #4 (see *Thrust #4: Monitoring & Control*).

System Software

HEC (and particularly HPC) systems have historically been performance driven. Most HEC systems use Linux, an operating system intended for servers or desktop scientific computing. Strides have been made to improve the performance of Linux for HEC systems over the past decade but the reliability is still largely untouched. At extreme scale the HPC community is often turning to light-weight operating systems (Cray's Catamount and IBM's CNK, for example) to improve performance.

Fundamentally very little has been done to harden operating systems and provide a robust set of interfaces to identify faults, communicate errors, and circumvent failures. The predominant mode of operation has remained to simply abort in the event of a problem. The resilience community could greatly benefit from standardization efforts for interfaces to better handle errors and provide means to continue correct operation, possibly in a degraded mode.

Middleware, Libraries, and APIs

Today, the line between libraries and programming models has blurred sufficiently that there is considerable overlap. Many application scientists rely on scientific libraries for highly tuned, optimized algorithms. Examples include mathematical libraries that provide linear algebra routines, data format libraries (such as HDF), and compression libraries. These types of libraries are highly tuned for performance, but historically not for reliability. There are many opportunities for the data integrity to be ensured at the library level entirely transparent to the user. Additionally there is a need for novel approaches that address hard errors at the library level.

Checkpointing libraries, on the other hand, are specifically intended to address application reliability and fault tolerance in the presence of hard failures. These libraries could benefit significantly from approaches that ensure the data integrity of checkpoints so that when applications "roll back" to a previous state there is high confidence that the state being reloaded has not been corrupted.

Tools

While a wealth of tools exist and continue to be developed to analyze the performance of an application of a platform, there exists very little in the way of resilience-related tools. Predominately the tools available today are used by system administrators to monitor the "health" of a system. These often look at metrics like voltages, fan speeds, temperature, and various statistics related to the resource manager (queue depth, wait time, average turnaround time). Experts are left to interpret this data and piece together an overall big picture of the system.

There exists a large gap between the application users and the system administrations with how they view the stability, usefulness, and success of HEC systems. Existing tools are not adequately capturing the "user experience" and cannot account for application stalls, hangs, transient errors, and abnormal terminations. Not only is there a need for these tools but there is further need for tools that build upon these to mine the data to determine the reliability of deployed systems. This information might be used to set optimal checkpoint intervals, drive purchasing policy, and generate more accurate application utilization graphs.

Cooperation and Coordination Frameworks

HEC resilience requires coordination among multiple layers (rather than improvement in or insertion of a single layer) [88, 89]. Currently today very few pieces of the software stack coordinate a response to a perceived fault. For example, if a node were to crash where a large application was running one might expect: the MPI layer would notice this, request from the resource allocator an additional node in the allocation, transparently restart the failed process on the new allocation, reload from the most recent checkpoint, and resume the computation – all entirely transparent to the application. Today, not only does this not exist but it cannot exist without standardized interfaces for brokering this type of interaction with different subsystem components.

Thrust #3: Fault Prediction & Detection

Fault prediction and detection are fundamental to HEC resilience, and there are many challenges to doing both at scale. In particular, the distinction between platform and application errors must be better understood in order to focus attention on those system faults that actually lead to application errors, as opposed to faults that do not impact the performance or integrity of the computation, as illustrated in Figure 3. More research is also needed to understand the implications of error latency, the variable and immeasurable duration of time between fault activation and failure, for both prediction and detection.

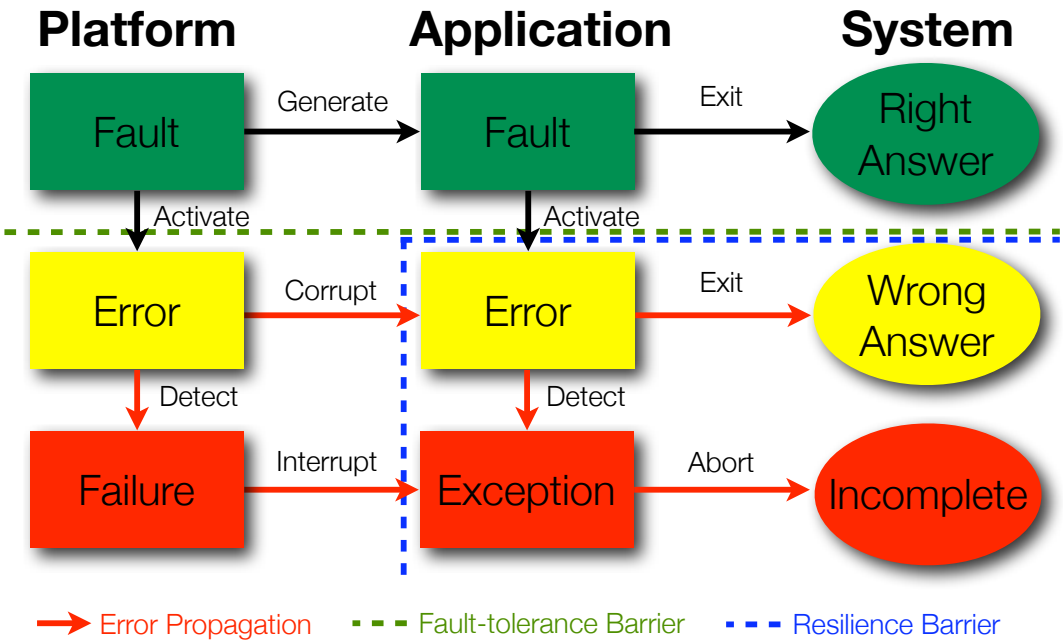

Figure 3. Comparing the resilience approach to that of traditional fault tolerance in terms of the progression of faults to failures on a *system*. Assume *fault*, *error*, and *failure* (*exception*) "events" are associated with both the *application* and the *platform* on which it runs. Since exit status for the *system* is the same for *right* and *wrong answers*, fault tolerance forms a barrier to prevent faults in either the *platform* or *application* from activating. Resilience, on the other hand, focuses exclusively on protecting the *application*.

Most contemporary failure prediction techniques involve SMTBF approximation and post-event analysis of system logs [90, 91]. On-line analysis based on individual compute node reliability in conjunction with system health monitoring is needed for near-real-time reliability awareness of HEC system resource and runtime system management.

In light of this, the goals of accurate fault prediction and detection for the system should be: (1) to minimize the time elapsed between the activation of a fault and the detection of the resultant error; and (2) to maximize the time elapsed between the prediction of a fault and subsequent activation. Research is needed in both of these areas, with an emphasis on

quantification of the rates of false positives and false negatives and with strategies to minimize those false indicators.

Statistical Analysis

Given that large-scale systems are composed of large numbers of components that are intended to be identical within manufacturer tolerances, statistical analysis is recognized as a promising technique for failure prediction and detection. Application of such techniques in this area is not straightforward as each component is subject to different physical and computational environments (e.g, running different code). Additionally, given the number of components, the number of possible attributes to be compared, and the frequency of data collection, scalable techniques are required. Further, the relationships of events in time must be considered, and noise and uncertainty in the measurement of quantities associated with the system's state must be handled.

Research areas include: scalable methodologies, multi-variate methodologies, time-series and event correlation methodologies and techniques for handling noise and non-uniform backgrounds, data fusion of textual and numerical data, uncertainty quantification.

Machine Learning

As a system is collecting data on application and platform performance, the resilience infrastructure should have the ability to respond to known fault indicators and recognize new patterns of system anomalies that indicate a degraded system state that have the potential to impact the end-to-end integrity of a computation. Research is needed in ways to best apply techniques of machine learning to recognize and categorize patterns of failure quickly and efficiently from a very large and complex set of indicators. The machine learning problem becomes a data intensive computing problem with the end goal being a capability for doing highly accurate, real time anomaly detection.

Anomaly Detection

Similar to statistical analysis, anomaly detection is recognized as a promising technique, since components are expected to behave alike under the same usage models. In addition to the issues mentioned above, anomaly detection suffers from the additional impediment that the events of interest in the test data (e.g., failures) are rare and hence drawing conclusions about causal relationships is difficult.

Some of the research areas include: statistical techniques, building useful classifiers using sparse or incomplete data and various techniques of machine learning.

Visualization

Visualization can be a powerful tool for identifying both regularities and irregularities in data sets. Since this work targets large-scale systems, visualization techniques are required that can represent large-scale systems and related data at varying degrees of fidelity as well as enabling techniques in large-scale distributed real-time rendering.

Research areas include: large-scale visualizations, varying fidelity visualizations, data fusion visual representations including both cross component and various types (e.g., numerical and textual), multiple time-series visualization, distributed real-time rendering.

Data and Information Collection

In order to determine when a failure has occurred, or is anticipated to occur, the detection and prediction mechanisms require data. The primary difficulty with data and information collection at extreme scale is the shear volume of platform and application data available. Research is required to find better ways to prune data, to collect only data which is helpful in the prediction and detection process, and to find ways to quickly and scalably collect and store the required data without adversely impacting system performance. To that end, both in-band and out-of-band data collection techniques should be examined, along with the hardware and software tools required to perform the collection.

Thrust #4: Monitoring & Control

The topic area of monitoring and control is primarily concerned with determining the current state of the system (hardware and software) and manipulating that state to maintain reliable operation. One of the fundamental principles of control theory is that of observability and controllability (i.e., we cannot hope to control systems that we cannot adequately observe). Today, this is a large problem in HEC and particularly HPC systems. Systems have grown to such incredible scale, determining the state of the system has become very difficult. Today, determining answers to simple questions "is the system up?", "is it stable?", "is it performing correctly?", is difficult. Tomorrow, more complex questions must be answered to achieve system resilience.

The monitoring and control thrust area covers topics that address this question. These include, but are not limited to, RAS systems, tunable fidelity, quality of service, and performability.

RAS Systems

RAS systems provide a critical foundation for many, if not all, resilience related efforts. To meet the goals of HEC the responsibilities shouldered by RAS systems will increase exponentially. To meet this challenge, improvements to existing RAS systems must be accomplished in the following areas:

- Component Level Hardware Interfaces – Either by leveraging commodity offerings in novel ways or by developing new standards in this area, low level sensor and monitoring interfaces that can maximize out of band monitoring, control and interaction with platform nodes and other components is essential.
- RAS System Hardware Architecture – Investigation into hardware architectures that can meet the enormous scale of next generation systems is required to meet increasing requirements of RAS systems.
- RAS System Software Architecture – New system software architectures must be developed that can both leverage evolving RAS system hardware architectures and

provide ubiquitous platform and application information to all stake holders from a system level standpoint (see *Systems Software & Middleware*, below)

- RAS Communication Protocols – To achieve the scalability required by future RAS systems, investigation into new communication protocols appropriate for RAS system communication is necessary.
- Failure and Resilience Methods – While the primary goal of a RAS system is to enhance the fault tolerance and resilience of the platform it serves, the scale of RAS systems required to support future HEC platforms presents a resilience problem of its own. New fault tolerance methodologies must be developed to enable RAS systems to meet their own Resilience needs (see *Systems Software & Middleware*, below).

In combination or individually these issues must be investigated to achieve resilience for future HEC systems.

Standards & Standard Framework for Monitoring and Control

Information about, not limited to, the platform (hardware), scheduling system, runtime system and the application is critical to all aspects of resilience. This information, when available, is often stored and accessible using widely disparate methods. To achieve the increasing demand for access to information necessary to meet future resilience goals a standard method of information mining must be developed.

This interface (possibly an Application Programming Interface API) must standardize the way in which an increasing number of resilience stake holders can access a growing amount of information about HEC systems, enabling progress in all resilience thrust areas. While lower level RAS systems may be vendor specific, this standard would facilitate a system independent method of obtaining all available information pertaining to the goal of the inquiry in a system independent manner.

System Software & Middleware

RAS systems software has traditionally been vendor specific and narrowly targeted to predominantly hardware health monitoring and system control. Future HEC systems will require significant advances to RAS system software to meet the wide spectrum requirements implied by resilience needs. From a broad perspective, advances to the overall RAS system software architecture are necessary to address critical scalability challenges. A more focused approach might target RAS system communication protocols and the specific requirements of information sharing, dispersion and retention.

Additionally, new fault tolerance methodologies that address the unique needs of HEC RAS systems must be developed. The growing numbers of RAS hardware components alone threaten to impact the reliability of future HEC RAS systems much like component reliability affects current HEC systems. While some challenges are similar, the specific purpose served by RAS systems may allow creative approaches to resilience not practical for the HEC system itself.

Tunable Fidelity

Tunable fidelity deals with the notion that a system should be able to be configured so that a varying level of reliability is provided in exchange for something else. Most commonly this exchange would be for performance or power consumption. The concept of tunable fidelity is relatively foreign in today's HPC systems and many HEC systems as well. Most scientists are uncomfortable with the idea that a computer system would give them anything but a perfect answer to as many decimal places as they ask. There is need, however, for research into algorithms and the supporting infrastructure that allow applications to adjust these "knobs".

One could envision programming language extensions that, based on observed conditions or programmer specification, turned down the accuracy of a calculation for a portion not deemed to need a high degree of accuracy while then increasing it for critical portions. System software and hardware could automatically put the system into a degraded mode (control) based on observed conditions (monitor). One simple example of this currently is the Cell BE processor where SPUs that are overheating (due to proximity to cache, for instance) can downclock the entire set of SPUs on a chip. It might not be cost effective to tune this fidelity in complex hardware and software environment. One possible solution would be to add ECC to all data being processed within the environment. The amount of ECC would be defined by the user and be a function of the confidence that is needed for the results of the computation. This method could be used to archive data on disk tuning the fidelity of the ECC based on the importance of the data.

Quality of Service

There exist very few QoS metrics on HEC systems of today. As mentioned earlier (see *Thrust #2: Enabling Infrastructure – Tools*), we do not currently have good measures of the user experience on a HEC system and it very often differs from platform monitoring observations. As such, there is a need for research in how to define, determine, and directly control the QoS of a given system.

Performability

The HEC community is very familiar with performance models that are used to predict and model the performance of theoretical and existing systems. Today, there is effort in creating coupled performance / power models. Similarly, the resilience community has a need for models that include the reliability of these systems. Performability is the term used to describe this coupling of performance and reliability and today there exists no good models that can be used for this purpose. System designers need to ask questions like "how might adding phase change memory to each node in my system effect a given application's performance and that application's reliability?"

More far reaching research is needed in the coupling of performance, reliability, and power. We foresee a need for models and simulations for these purposes.

Thrust #5: End-to-End Data Integrity

Data integrity is essential for resilient computing and underpins all aspects of high-end computing. HEC systems are used to simulate real world phenomena, predict events, and are critical to national security. It is imperative to have confidence in getting the right answer and using correct data to make informed decisions. A sub-category of data integrity deals with the potential malicious modification of data.

There are numerous techniques to ensure data integrity in the face of unreliable components that have been developed and are available in the literature. Table 2 is one way to categorize the wide field of data integrity. One observation that comes out of this taxonomy is that data integrity can be ensured at many different levels of the system stack. In fact, hybrid approaches where data integrity is assured at different levels of the system stack in different ways may lead to the most effective solutions.

		Covers Entire System	Covers Multiple Components	Covers Single Component	Covers Multiple Structures	Covers Single Structure/ Multiple Bits	Covers Single Bit
Software Level	Application	Software RMT, API level checks, Self-checking code					
	Operating System	Bounds Checks					
Board Level	Architectural		Socket-level loose lock-step	Redundancy – lock-step, system RMT			
	Electrical		CRC checks on all board level busses & interfaces				
Component Level	Architectural/ Micro-architectural				RMT, residue, periodic flushing	Redundancy, Parity, ECC, Scrubbing	
	Circuit						Rad-hard circuits (many different flavors)
	Manufacturing Process			Fully depleted, SOI, Triple-well			

Table 2. A taxonomy of techniques for maintaining data integrity

As microprocessors become more and more complex as a result of ever-increasing device densities and the thirst for higher performance (whether it be single-threaded performance in the form of more sophisticated core architectures, wider execution units and SIMD structures or multi-threaded performance in the form of multiple cores and higher memory bandwidth), the vulnerability to errors, both hard and soft, increases as well. As a result, data integrity in some form at the hardware level is a necessity. As seen

in the taxonomy, there exists in the literature today many such techniques that can be implemented at the hardware level. Each of these solutions comes with a cost in terms of die area, power, and in some cases performance.

In modern microprocessors, there is an additional dimension to these costs and that is the cost of development, which includes both design effort and validation time. As microprocessor complexity increases there are more and more bits in the form of additional sequentials or micro-architectural structures that must be protected against errors. As the number of vulnerable bits grows, end-to-end error mitigation solutions become far more attractive than per-bit or per-structure solutions. End-to-end data integrity schemes tend to have the ability to protect multiple bits and/or structures, including entire data and control paths with a single mechanism. This is essential in keeping the cost of development reasonable, both in terms of manpower and time.

One related question facing us is whether to pay the premiums involved in developing a custom hardware solution or to accept a lower level of hardware data integrity, perhaps by applying other data integrity solutions at higher levels of the system stack to make up the slack at the hardware level, in order to reap the cost benefits using of higher volume, general purpose components.

Silent Data Corruption

Silent data corruption, SDC, poses a threat to computational science, with several studies documenting SDC or related problems on real systems [92, 93, 94, 95, 96]. As such it is imperative that research be undertaken to characterize the impact of SDC on users and that methods to protect computations from SDC be developed.

While information in the public domain is still relatively scarce, there is some emerging knowledge. For example, SDC can have multiple causes, e.g. temperature/voltage fluctuations, particles, manufacturing residuals, oxide breakdown and electrostatic discharge [21], and will likely be more prevalent in new technologies [20, 97, 98]. From a statistical perspective, for a given device susceptibility, a platform containing more replicates of the device is more likely to be affected than one containing fewer replicates. It is also possible that SDC could affect desktop computers and laptops, with the laptops and desktops used for scientific computation at an institution perhaps equivalent to a cluster [99].

Thus, from an operational perspective there is an urgent need for answers to questions such as the following:

- What is the probability that a code that runs for *h* hours on *n* nodes on a particular platform gets the wrong answer? How does this depend on which code is running?
- What is the probability that if I write my data to disk and then read it back again, I get the same results?
- What operational strategies can mitigate the impact of SDC on users?

Answering these operational questions will require fundamental research focused on both characterizing SDC and the faults that can lead to SDC and resilience methods for mitigating the impact of SDC on applications.

Conclusions

There is an pressing need for more focused and complete government investment in the area of HEC Resilience, given the growing impact of errors on today's extreme scale systems and the projected trends for the systems of tomorrow. Resilience has henceforth received too little attention and funding in the research community, as compared to other elements of HEC. This deficiency is at least in part because of the magnitude of the challenges to be overcome and the difficulty in making progress in any one aspect of the problem without simultaneous advances in the other key thrust areas. The wide range of basic research that must be coordinated over the multiple thrust areas that make up the field of HEC resilience in order to make meaningful progress complicates the challenge.

Thus we find that from resilience computing theory to the enabling infrastructure that employs it and from accurate prediction and detection of faults to monitoring and control of the systems encountering those faults there exists a critical need for resilience research. And finally, there is the underpinning of data integrity, which impacts all HEC systems and threatens to undermine the integrity of the information, derived from HEC data that drives government policy and protects national security. In order for our nation to make effective use of high-end computing in the next decade and maintain crucial national preeminence in the field of HEC there is little doubt that resilience concerns must become a research priority. A coordinated, multi-thrust approach of basic research, tool development and data analysis as outlined in this report will be required to meet the challenge.

References

[1] Report of the High End Computing Revitalization Task Force (HECRTF), May, 2004.

[2] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. In Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2007, volume 78, pages 2022–2032, Boston, MA, USA, June 24-28, 2007. Institute of Physics Publishing, Bristol, UK. URL http://www.iop.org/EJ/abstract/1742-6596/78/1/012022.

[3] J. T. Daly. "Application Resilience for Truculent Systems" presented at the 2009 Fault Tolerance Workshop for Extreme Scale Computing, http://www.teragridforum.org/mediawiki/images/8/80/Daly2009ws.pdf.

[4] "Roadrunner to Expand Hybrid Computing Applications", ASCeNews Quarterly News Letter, NA-ASC-500-09—Issue 10.

[5] "Testing & Integration", Petascale Systems Integration Into Large Scale Facilities Workshop, May 15-16, 2007, http://www.nersc.gov/projects/HPC-Integration/presentations/Breakout_3_Testing_Integration.ppt.[

[6] http://www.top500.org/

[7] http://www.nccs.gov/jaguar/

[8] http://www.exascale.org/mediawiki/images/a/a1/Iesp-roadmap-draft-0.93-complete.pdf, section 3.1, page 6.

[9] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, and S. L. Scott. Using log information to perform statistical analysis on failures encountered by large-scale HPC deployments. In Proceedings of the 2008 High Availability and Performance Computing Workshop, 2008.

[10] M. Seager. Operational machines: ASCI White. Talk at the 7th Workshop on Distributed Supercomputing (SOS) 2003, Mar. 4-6, 2003.

[11] C.-H. Hsu and W.-C. Feng. A power-aware run-time system for high-performance computing. In Proceedings of the IEEE/ACM International Conference on High Performance Computing and Networking (SC) 2005, Seattle, WA, USA, Nov. 12-18, 2005.

[12] National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, USA. Current and past HPC system availability statistics, 2007.

[13] J. Morrison. The ASCI Q system at Los Alamos. Talk at the 7th Workshop on Distributed Supercomputing (SOS) 2003, Mar. 4-6, 2003.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29–43, 2003.

[15] F. H. Streitz. Simulating solidification in metals at high pressure – The drive to petascale computing. Keynote address at the 12th Annual San Diego Supercomputer Center (SDSC) Summer Institute 2006, July 17-21, 2006.

[16] F. H. Streitz. Simulating solidification in metals at high pressure – The drive to petascale computing. Keynote address at the 12th Annual San Diego Supercomputer Center (SDSC) Summer Institute 2006, July 17-21, 2006.

[17] H. Song, C. Leangsuksun, and R. Nassar. Availability modeling and analysis on high performance cluster computing systems. In First International Conference on Availability, Reliability and Security, pages 305–313, 2006.

[18] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3):382–401, 1982. ISSN 0164-0925. URL http://doi.acm.org/10.1145/357172.357176.

[19] Standards Coordinating Committee 10 (Terms and Definitions) Jane Radatz (Chair). The IEEE Standard Dictionary of Electrical and Electronics Terms, volume IEEE Std 100-1996. IEEE Publishing, 1996.

[20] S. Borkar. "Major Challenges to Achieve Exascale Performance," 2009 Salishan Conference on High-Speed Computing, 2009. URL http://www.lanl.gov/orgs/hpc/salishan/index09.shtml

[21] C. Constantinescu. "Silent Data Corruption: Causes and Mitigation Techniques," Invited Talk presented at Los Alamos National Laboratory October 9, 2008.

[22] Cray Inc. The Cray X-MP series of computer systems, 1985. URL http://archive.computerhistory.org/resources/text/Cray/Cray.X-MP.1985.102646183.pdf.

[23] S. G. Tucker. The IBM 3090 system: An overview. IBM Systems Journal, 25(1):4–19, 1986. URL http://www.research.ibm.com/journal/sj/251/tucker.pdf.

[24] R. Berrendorf, H. C. Burg, R. Esser, M. Gerndt, and R. Knecht. Intel paragon xp/s - architecture, software environment, and performance. Number KFA-ZAM-IB-9409, Julich, Germany, 1994. URL http://www.fz-juelich.de/zam/docs/printable/ib/ib-94/ib-9409.ps.

[25] A. J. van der Steen and J. J. Dongarra. Overview of recent supercomputers, 2001. URL http://www.phys.uu.nl/˜steen/web01/overview01.html.

[26] IBM Corporation, Armonk, NY, USA. IBM Blue Gene computing platform documentation, 2007. URL http://www-03.ibm.com/servers/deepcomputing/bluegene.html.

[27] Cray Inc., Seattle, WA, USA. Cray XT5 computing platform documentation, 2009. URL http: //www.cray.com/products/xt5.aspx.

[28] National Center for Atmospheric Research, Boulder, CO, USA. Community Climate System Model (CCSM) documentation, 2007. URL http://www.ccsm.ucar.edu.

[29] M. Schulz, G. Bronevetsky, R. Fernandes, D. Marques, K. Pingali, and P. Stodghill. Implementation and evaluation of a scalable application-level checkpoint-recovery scheme for MPI programs. In Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking and Storage (SC) 2004, Pittsburgh, PA, USA, Nov. 6-12, 2004. IEEE Computer Society. ISBN 0-7695-2153-3. URL http://dx.doi.org/10.1109/SC.2004.29.

[30] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. IEEE Transactions on Parallel and Distributed Systems (TPDS), 9(10):972–986, 1998. ISSN 1045-9219. URL http://doi.ieeecomputersociety.org/10.1109/71.730527.

[31] G. Stellner. CoCheck: Checkpointing and process migration for MPI. In Proceedings of the 10th IEEE International Parallel and Distributed Processing Symposium (IPDPS) 1996, Honolulu, HI, USA, Apr. 15-19, 1996. IEEE Computer Society. URL http://doi.ieeecomputersociety.org/10.1109/IPPS.1996.508106.

[32] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical Report CS-TR-199701346, Computer Sciences Department, University of Wisconsin Madison, Madison, WI, USA, 1997. URL http://www.cs.wisc.edu/condor/doc/ckpt97.ps.

[33] P. H. Hargrove and J. C. Duell. Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters. In Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2006, volume 46, pages 494–499, Denver, CO, USA, June 25-29, 2006. Institute of Physics Publishing, Bristol, UK. URL http://www.iop.org/EJ/article/1742-6596/46/1/067/jpconf6_46_067.pdf.

[34] S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. In Proceedings of the Los Alamos Computer Science Institute (LACSI) Symposium 2003, Santa Fe, NM, USA, Oct. 27-29, 2003. URL http://ftg.lbl.gov/CheckpointRestart/lacsi-2003.pdf.

[35] Lawrence Berkeley National Laboratory, Berkeley, CA, USA. Berkeley Lab Checkpoint/Restart (BLCR) documentation, 2007. URL http://ftg.lbl.gov/checkpoint.

[36] J. M. Squyres and A. Lumsdaine. A component architecture for LAM/MPI. In Lecture Notes in Computer Science: Proceedings of the 10th European PVM/MPI Users' Group Meeting (EuroPVM/MPI) 2003, volume 2840, pages 379–387, Venice, Italy, Sept. 29 - Oct. 2, 2003. Springer Verlag, Berlin, Germany. ISBN 978-3-540-20149-6. URL http://www.lam-mpi.org/papers/euro-pvmmpi2003/euro-pvmmpi-2003.pdf.

[37] Indiana University, Bloomington, IN, USA. Local Area Multicomputer Message Passing Interface (LAM-MPI) documentation, 2007. URL http://www.lam-mpi.org.

[38] J. Hursey, T. I. Mattox, and A. Lumsdaine. Interconnect agnostic checkpoint/restart in Open MPI. In Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC) 2009, pages 49–58, Munich, Germany, June 11-13, 2009. ACM Press, New York, NY, USA. ISBN 978-1-60558-587-1. URL http://doi.acm.org/10.1145/1551609.1551619.

[39] L. M. Silva and J. G. Silva. An experimental study about diskless checkpointing. In Proceedings of the 24th Euromicro Conference on Engineering Systems and Software for the Next Decade 1998, page 10395, Vesteras, Sweden, Aug. 25-27, 1998. IEEE Computer Society. ISBN 8186-8646-4-1. URL http://doi.ieeecomputersociety.org/10.1109/EURMIC.1998.711832.

[40] G. Zheng, L. Shi, and L. V. S. Kale. FTC-Charm++: An in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In Proceedings of the 6th IEEE International Conference on Cluster Computing (Cluster) 2004, pages 93–103, San Diego, CA, USA, Sept. 20-23, 2004. IEEE Computer Society. ISBN 0-7803-8694-9. URL http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2004.1392606.

[41] G. Bronevetsky and A. Moody. Scalable I/O systems via node-local storage: Approaching 1 TB/sec file I/O. Technical Report TR-JLPC-09-01, Lawrence Livermore National Laboratory, Livermore, CA, USA, Aug. 2009. URL http://dx.doi.org/10.2172/964079.

[42] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA, Nov. 1994. ISBN 978-0-262-57108-1. URL http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=6689.

[43] A. Agbaria and R. Friedman. Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In Proceedings of the 8th International Symposium on High Performance Distributed Computing (HPDC)

1999, pages 31–40, Redondo Beach, CA, USA, Aug. 3-6, 1999. IEEE Computer Society. ISBN 0-7695-0287-3. URL http://doi.ieeecomputersociety.org/10.1109/HPDC. 1999.805295.

[44] K. P.Birman, B. Constable, M. Hayden, J.Hickey, C. Kreitz, R. van Renesse, O. Rodeh, and W. Vogels. The Horus and Ensemble projects: Accomplishments and limitations. In Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX) 2000, volume 1, pages 149–161, Hilton Head, SC, USA, Jan. 25-27 2000. IEEE Computer Society. ISBN 0-7695-0490-6. URL http://www.cs.cornell.edu/projects/quicksilver/public_pdfs/Horus%20and%20Ensemble.pdf.

[45] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. Fault-tolerant MPI for the Harness metacomputing system. In Lecture Notes in Computer Science: Proceedings of the 1st International Conference on Computational Science (ICCS) 2002, Part I, volume 2073, pages 355–366, San Francisco, CA, USA, May 28-30, 2001. Springer Verlag, Berlin, Germany. URL http://www.netlib.org/utk/people/JackDongarra/PAPERS/ft-harness-iccs2001.ps.

[46] Message passing interface (MPI) Forum. MPI 3.0 Fault Tolerance Working Group, 2009. URL http://svn.mpi-forum.org/trac/mpi-forum-web/wiki/FaultToleranceWikiPage.

[47] E. N. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. IEEE Transactions on Computers (TC), 41(5):526–531, 1992. ISSN 0018-9340. URL http://dx.doi.org/10.1109/12.142678.

[48] S. Rao, L. Alvisi, and H. V. Vin. Egida: An extensible toolkit for low-overhead tault-tolerance. In Proceedings of the 29th IEEE International Symposium on Fault-Tolerant Computing (FTCS) 1999, pages 48–55, Madison, WI, USA, June 15-18 1999. IEEE Computer Society. ISBN 0-7695-0213-X. URL http://doi.ieeecomputersociety.org/10.1109/FTCS.1999.781033.

[49] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello. MPICH-V: A multiprotocol fault tolerant MPI. International Journal of High Performance Computing and Applications (IJHPCA), 20(3):319–333, 2006. ISSN 1094-3420. URL http://mpich-v.lri.fr/papers/ijhpca_mpichv.pdf.

[50] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems (TOCS), 3(1):63–75, 1985. ISSN 0734-2071. URL http://doi.acm.org/10.1145/214451.214456.

[51] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. IEEE Transactions on Computers (TC), C-33(6):518–528, 1984. ISSN 0018-9340. URL http://dx.doi.org/10.1109/TC.1984.1676475.

[52] Z. Chen and J. J. Dongarra. Algorithm-based checkpoint-free fault tolerance for parallel matrix computations on volatile resources. In Proceedings of the 20st IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2006, page 10, Rhodes Island, Greece, Apr. 25-29, 2006. IEEE Computer Society. ISBN 1-4244-0054-6. URL http://icl.cs.utk.edu/news_pub/submissions/checkpoint_free.pdf.

[53] C. Engelmann and G. A. A. Geist. Super-scalable algorithms for computing on 100,000 processors. In Lecture Notes in Computer Science: Proceedings of the 5th International Conference on Computational Science (ICCS) 2005, Part I, volume 3514, pages 313–320, Atlanta, GA, USA, May 22-25, 2005. Springer Verlag, Berlin, Germany. ISBN 978-3-540-26032-5. URL http://www.csm.ornl.gov/~engelman/publications/engelmann05superscalable.pdf.

[54] D. Chazan and M. Miranker. Chaotic relaxation. Linear Algebra and its Applications, 2:199–222, 1969.

[55] H. Ltaief, E. Gabriel, and M. Garbey. Fault tolerant algorithms for heat transfer problems. Journal of Parallel and Distributed Computing (JPDC), 68(5):663–677, 2008. ISSN 0743-7315. URL http://dx.doi.org/10.1016/j.jpdc.2007.09.004.

[56] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In Proceedings of the 21st ACM International Conference on Supercomputing (ICS) 2007, pages 23–32, Seattle, WA, USA, June 16-20, 2007. ACM Press, New York, NY, USA. ISBN 978-1-59593-768-1. URL http://www.csm.ornl.gov/~engelman/publications/nagarajan07proactive.pdf.

[57] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive process-level live migration in HPC environments. In Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2008, Austin, TX, USA, Nov. 15-21, 2008. ACM Press, New York, NY, USA. ISBN 978-1-4244-2835-9. doi: http://doi.acm.org/10.1145/1413370.1413414. URL http://www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf.

[58] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2007, pages 1–12, Reno, NV, USA, Nov. 15-21, 2007. ACM Press, New York, NY, USA. ISBN 978-1-59593-764-3. URL http://doi.acm.org/10.1145/1362622. 1362678.

[59] J. Stearley and A. J. Oliner. Bad words: Finding faults in Spirit's syslogs. In Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid) 2008: Workshop on Resiliency in High Performance Computing (Resilience) 2008, Lyon, France, May 19-22, 2008. IEEE Computer Society. URL http://xcr.cenit.latech.edu/resilience2008/program/resilience08-3.pdf.

[60] J. Hofer and T. Fahringer. Synthesizing Byzantine fault-tolerant grid application wrapper services. In Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CC-Grid) 2008, pages 467–474, Lyon, France, May 19-22, 2008. IEEE Computer Society. ISBN 978-0-7695-3156-4. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=4534182&arnumber=4534251&count=129&index=68.

[61] R. M. Russell. The CRAY-1 computer system. Communications of the ACM, 21(1):63–72, 1978. ISSN 0001-0782. URL http://doi.acm.org/10.1145/359327.359336.

[62] S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender. Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer. IEEE Transactions on Device and Materials Reliability (TDMR), 5(3):329–335, 2005. ISSN 1530-4388. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1545893.

[63] Advanced Micro Devices, Inc., Sunnyvale, CA, USA. AMD Opteron Processor Product Data Sheet, 2007. URL http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf.

[64] A. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In Proceedings of the 21th ACM International Conference on Supercomputing (ICS) 2007, Seattle, WA, USA, June 16-20, 2007.

[65] H. Song, C. Leangsuksun, and R. Nassar. Availability modeling and analysis on high performance cluster computing systems. In First International Conference on Availability, Reliability and Security, pages 305–313, 2006.

[66] S. Rani, C. Leangsuksun, A. Tikotekar, V. Rampure, and S. Scott. Toward efficient failure detection and recovery in HPC. In High Availability and Performance Computing Workshop, 2006.

[67] A. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In Proceedings of the 21th ACM International Conference on Supercomputing (ICS) 2007, Seattle, WA, USA, June 16-20, 2007.

[68] I. R. Philp. Software failures and the road to a petaflop machine. In Proceedings of the 1st Workshop on High Performance Computing Reliability Issues (HPCRI) 2005, in conjunction with the 11th International Symposium on High Performance Computer Architecture (HPCA) 2005, San Francisco, CA, USA, Feb. 12-16, 2005. IEEE Computer Society.

[69] J. Fabula, J. Moore, and A. Ware. Understanding neutron single-event phenomena in FPGAs. Military Embedded Systems, 3(2), 2007. ISSN 1557-3222. URL http://www.mil-embedded.com/pdfs/Xilinx.Mar07.pdf.

[70] T. Heijmen. Radiation-induced soft errors in digital circuits – A literature survey. Technical Report 2002/828, Philips Electronics, The Netherlands, 2002.

[71] F. L. Kastensmidt, L. Carro, and R. Reis. Fault-Tolerance Techniques for SRAM-based FPGAs, volume 32 of Frontiers in Electronic Testing. Springer Verlag, Berlin, Germany, 2006. ISBN 978-0-387-31068-8. URL http://www.springerlink.com/content/q10336. [32] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3):382–401, 1982. ISSN 0164-0925. URL http://doi.acm.org/10.1145/357172.357176.

[72] S. Mukherjee. Architecture Design for Soft Errors. Morgan Kaufmann Publishers, Burlington, MA, USA, Feb. 2008. ISBN 978-0-12-369529-1. URL http://www.elsevier.com/wps/product/cws_home/713719.

[73] T. Heijmen, P. Roche, G. Gasiot, K. R. Forbes, and D. Giot. A comprehensive study on the soft-error rate of flip-flops from 90-nm production libraries. IEEE Transactions on Device and Materials Reliability (TDMR), 7(1):84–96, 2007. ISSN 1530-4388. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4271495.

[74] K. Kanoun and L. Spainhower. Dependability Benchmarking for Computer Systems. Wiley Inter-Science, John Wiley & Sons, Inc., Hoboken, NJ, USA, July 2008. ISBN 978-0-470-23055-8. URL http://www.wiley.com/WileyCDA/WileyTitle/productCd-047023055X.html.

[75] E. N. M. Elnozahy, R. Bianchini, T. El-Ghazawi, A.Fox, F. Godfrey, A.Hoisie, K. McKinley, R. Melhem, J. S. Plank, P. Ranganathan, and J. Simons. System resilience at extreme scale. Technical report, Defense Advanced Research Project Agency (DARPA), 2008. URL http://institutes.lanl.gov/resilience/docs/Toward%20Exascale%20Resilience.pdf.

[76] J. T. Daly. ADTSC nuclear weapons highlights: Facilitating high-throughput ASC calculations. Technical Report LALP-07-041, Los Alamos National Laboratory, Los Alamos, NM, USA, June 2007. URL http://www.lanl.gov/orgs/adtsc/publications/nw_highlights_2007/ch13/13_2daly_facilitating.pdf.

[77] J. T. Daly, L. A. Pritchett-Sheats, and S E. Michalak. Application MTTFE vs. platform MTTF: A fresh perspective on system reliability and application throughput for computations at scale. In Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid) 2008: Workshop on Resiliency in High Performance Computing (Resilience) 2008, Lyon, France, May 19-22, 2008. IEEE Computer Society. URL http://xcr.cenit.latech.edu/resilience2008/program/resilience08-10.pdf.

[78] J. Stearley. Defining and Measuring Supercomputer Reliabilty, Availability, and Serviceability. LCI Conference. 2005

[79] J. T. Daly. Performance Challenges for Extreme Scale Computing, SDI/LCS Seminar. 2007.

[80] Bell, Hack, et al. Advanced Scientific Computing Advisory Committee Petascale Metrics Report. 2007.

[81] R. Iyer, Z. Kalbarczyk, and M. Kalyanakrishnan. "Measurement-Based Analysis of Networked System Availability", Performance Evaluation, LNCS 1769, pp. 161-199, 2000.

[82] N. Singpurwalla and A. Wilson. "Probability, Chance, and the Probability of Chance". The IIE Transactions. Vol. 41, No. 1, pp. 12-22, 2008.

[83] G. Fagg and J. Dongarra. "Building and using a Fault Tolerant MPI implementation,"International Journal of High Performance Applications and Supercomputing, Vol.18, No.3, pp. 353-361, 2004.

[84] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth and Gengbin Zheng. "Programming Petascale Applications with Charm++ and AMPI". In Petascale Computing: Algorithms and Applications, pg 421-441, Chapman & Hall / CRC Press, ed D Bader, 2007.

[85] Chao Huang, Gengbin Zheng, Sameer Kumar and Laxmikant V. Kale. "Performance Evaluation of Adaptive MPI", Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006, March 2006.

[86] Chao Huang. "System Support for Checkpoint and Restart of Charm++ and AMPI Applications", MS Thesis, Dept. of Computer Science, University of Illinois, 2004.

[87] Gengbin Zheng, Lixia Shi and Laxmikant V. Kale. "FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI", 2004 IEEE International Conference on Cluster Computing, San Diego, CA, September, 2004. pp. 93-103.

[88] URL http://www.mcs.anl.gov/research/cifts/

[89] URL http://www.relxlayer.org/

[90] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. BlueGene/L failure analysis and prediction models. Dependable Systems and Networks, 2006. DSN 2006. International Conference on, pages 425–434, 2006.

[91] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1–12, New York, NY, USA, 2007. ACM.

[92] C. Constantinescu. "Teraflops Supercomputer: Architecture and Validation of the Fault Tolerance Mechanisms," IEEE Transactions on Computers 49:886-894, 2000.

[93] Kola et al. "Faults in Large Distributed Systems and What We Can Do About Them," Euro-Par 2005:442-453, 2005.

[94] Panzer-Steindel, "Data Integrity," 2007. URL http://storagemojo.com/2007/09/19/cerns-data-corruption-research/

[95] Bairavasundaram et al. "An Analysis of Data Corruption in the Storage Stack," Proceedings of the 6th USENIX Conference on File and Storage Technologies: 223-238, 2008.

[96] S. Michalak. "Silent Data Corruption Research at Los Alamos National Laboratory" presented as part of Invited Panel on ``Silent Data Corruption: Myth or Reality?" Dependable Systems and Networks, 2008.

[97] Pan et al. "A Low Cost Scheme for Reducing Silent Data Corruption in Large Arithmetic Circuits," Proceedings of the IEEE International Symposium of Defect and Fault Tolerance of VLSI Systems: 343-351, 2008

[98] C. Constantinescu (2006) "Intermittent Faults in VLSI Circuits," Proceedings of SELSE2 Workshop, 2006. URL http://selse2.selse.org/

[99] S. Michalak. "Silent Data Corruption: A Threat to Data Integrity in High-End Computing Systems" Invited Panelist at 2009 National HPC Workshop on Resilience, 2009.